

ActionScript 3 basics in plain speak, for those coming from earlier versions of Flash. And who still like to use Flash not Flex or some other tool. And who are not programmers.

Part 1

ActionScript 3 can exist within the Adobe Flash timeline, but not on buttons or movie clips.

In the past one could link to ActionScript (.AS) files on compiling – this has now become the more standard way to include code.

One can either link to an **AS file** (or multiple) or have code in the movie's timeline, but not the two methods combined. It is still permissible to have stops in the movie (or more likely the movie clips in the movie).

Remember – the **swf needs to be recompiled** (Test Movie or Publish) each time the code changes.

Also, keep in mind that in this new coding way means that an object that appears at 2 different frames with the same instance name will be seen as 2 distinct (separate) objects, causing code issues.

The Document Class.

This is the first ActionScript that the Flash file uses. If we were thinking html and style sheets, it is like redefining a tag, such as the body tag, and telling it what colour and size and font it should utilise. Objects in the movie – however you create them – will/may have their own classes. Again, think of html-orientated style sheets, where a custom class tells a specific bit of text what it should look like, or an object how it should behave.

The Structure of a Class.

```
package {  
  
    class FileName{  
  
        function FileName{  
  
            }//closes function  
  
        }//closes class  
  
    }//closes package
```

The explanation of this **basic version of the structure** follows.

What must be called what?

When I first started trying to get to grips with this, the first thing that mystified me was naming. In the help files the same name seemed to be used again and again.

When you create a new ActionScript file, save it with a name the pretty much describes – to you and other worker ants – what it will do. That just makes life easier.

Make sure it has a capital at the start – that is just convention.

In the above example of the structure, the AS file would be named "FileName".

The **Action Script file name**, the **class name** in the file and the **function name** that makes the class 'run' (or initialise) all must have the **SAME NAME**. The Flash file you are using it in can have **any** name.

What is a Package?

Each class file will have a **package** surrounding all the other code. **One** package will be allowed per AS file. Packages **cannot exist** in the Flash timeline. What do they do? Well, they are like a container that tells all the code **where it is stored**.

package {

Is an example of an unnamed package.

package CodeFolder {

Is an example of an AS file which lives in a folder called CodeFolder one directory away from the Flash file the AS file is linked to. So when we Test Movie, the Player goes into CodeFolder to find the code.

package CodeFolder.SubfolderA {

In this example, the AS files reside in a further sub-folder inside the CodeFolder. Think of addressing a movie clip that is nested – go into CodeFolder and then into SubfolderA and so on.

Which you choose to use is up to you, and the re-usability of the code you write. I started naming them and found no advantage in it, nor anyone who could offer a valid reason on-line, so I use unnamed packages. Meaning my ActionScript files lives in the same folder as my FLA files.

The Class and the Function

As mentioned before, the class really is a collection of code telling Flash Player what to do with the contents of the swf. Just the same as an 'old' frame script did. And in order to execute, it needs to start with a function of the same name. (Just like a function has to be called at some point in order to execute).

Is that all?

There are a few other standard elements we would find in the basic structure of a package. Things to import, and variables.

```
package {  
  
    import whatisinmovie  
  
    class FileName{  
  
        variables (global)  
  
        function FileName{  
  
            }//closes function  
  
        }//closes class
```

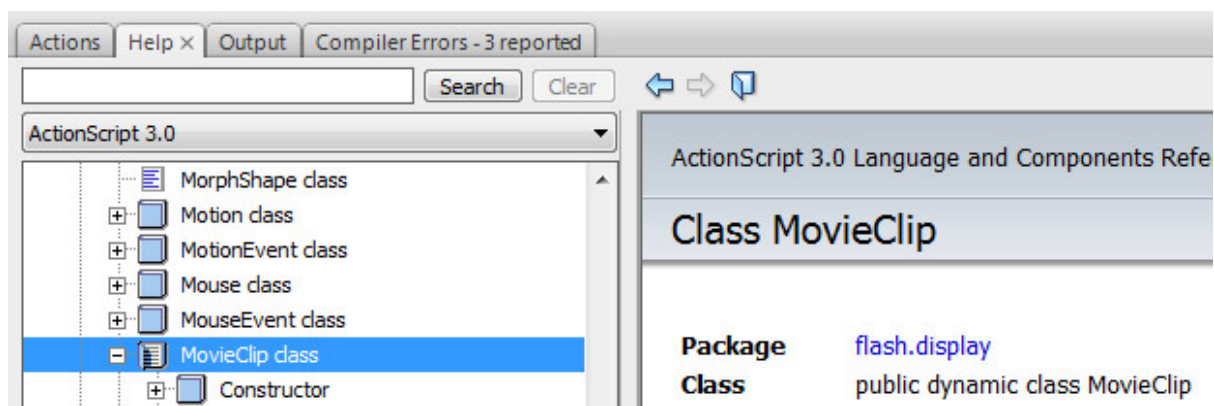
}//closes package

Import what?

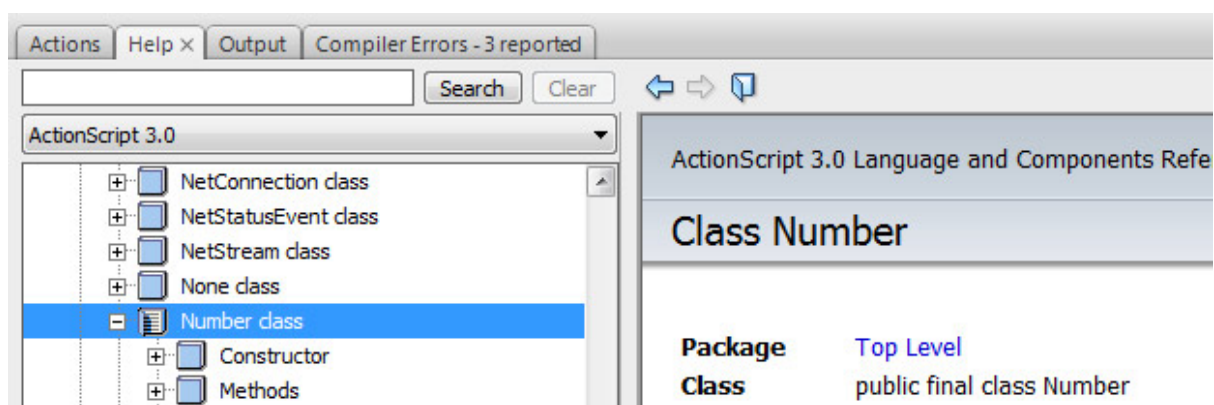
When you publish a Flash movie, it compiles the associated action script files as well as pre-made classes that do pre-set functions. Think of **stop** and **gotoAndPlay** as pre-made functions. And for those who have used **Macromedia Director**, think of how you add **Xtras** into a ‘stub’ projector, to ensure all media can be displayed. Or, if you have used **Tweening Classes**, think about how one has to import the tweening class first to use it.

So, if you test the swf, it will give you an error message like “1017: The definition of base class *MovieClip* was not found.”

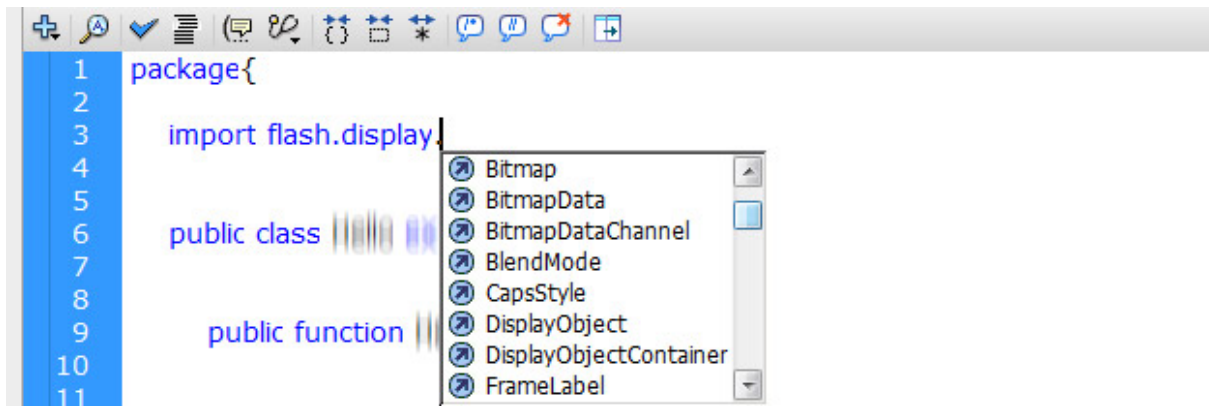
This indicates that, in this case, I need to import something or other to use a movie clip. How do I know what to import? Use the **help** file!



Ensure you are looking in ActionScript 3 reference. Select the type of object you are working with on the left. Click on the main ‘folder’ (one that says class). At the very top of the Help information on the right you will see which package is needed to use the item, in this case we need to import **flash.display**.



If it says “**Top Level**” this means this class is included by default when the movie is compiled, so nothing needs to be imported.



When you type into your AS file, prompts will appear after the dot you type, allowing you to select the specific class or package to import. Again, this is like a folder system, so we are importing all or some of the package content the more specific we are.

How much do I import?

I have read that it is best to import very specifically, rather than everything. I do not really note a size difference, so I think this is more a good practise thing.

Consider the following:

```
import flash.events.*;
```

This asterisks indicates to import ALL 'sub classes' of the events package.

VERSUS

```
import flash.events.MouseEvent.DOUBLE_CLICK;
```

I am importing only the double click event for a mouse click in the events package. Are you seriously going to import rollover, rollout, press, release etc all separately? I think not.

[An aside at this point: dragOver and dragOut events no longer exist]

So, be specific, but not **too** specific. See what works for you as you get more used to coding this method; I tend to import more than required in a lazy fashion.

```
import flash.display.*;
import flash.events.*;
import flash.net.*;
import flash.text.*;
```

The above represent my standard imports for a 'basic' Flash file, where I need to link to an html page or load something (hence net) and utilise text formatting and styling.

Extends or Error

Another error may appear due to our current structure: "5000: The class 'FileName' must subclass 'flash.display.MovieClip'" since it is linked to a library symbol of that type."

The main timeline is really a movie clip. So if we are using **Flash** (not Flex) as our IDE (*Integrated Development Environment*) we are by its very nature using a movie clip. So our class **MUST extend** the **MovieClip** (or **Sprite**) class. It takes the form of

```
class FileName extends MovieClip{
```

Here's an analogy. You want to get coffee. So we make a coffee class. But unless we have a mug we have little chances of successfully getting coffee. So we extend our coffee class to use mug as its **base**.

What's this Sprite thing?

Sprite is recommended as more 'advanced' to use than Movie Clip, for 'proper' coders, blah blah. (The blogs out there can be so arrogant at times).

Sprite has all the standard properties of a movie clip EXCEPT it only has/allows for **one frame**. If you are generating something by code, it may as well be a sprite, as you cannot create second or greater frames. So, if you have a single frame timeline with single frame movie clips – use Sprite. If you are planning on using movie clips from the library that animate, use MovieClip to extend the class or you will get an error.

And for those people out there – there is more to animation than tweening classes. The cleverness of a 'hand' animated movie clip cannot be beaten. Proper coding can too often make for a dull site.

Can we please make something now?

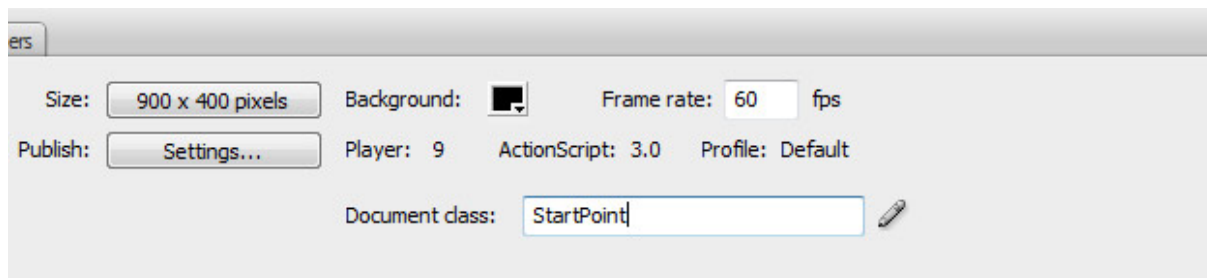
Yep, all theory makes Jack a very bored boy.

In Flash, make a new **Flash File** (ActionScript 3) and save it into a folder as "lesson1".

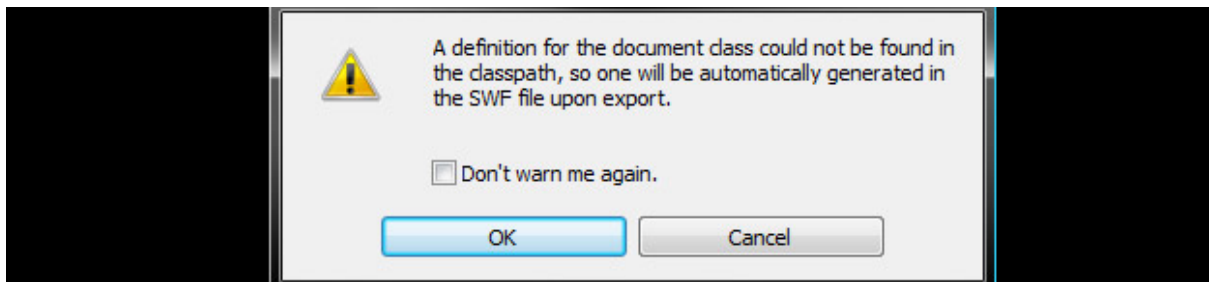
In Flash, make a new **ActionScript file**, save it into the same folder (our package will have no name) as "**StartPoint**".

I find it better to create and Save the 2 initial documents so that the linking will be free of errors.

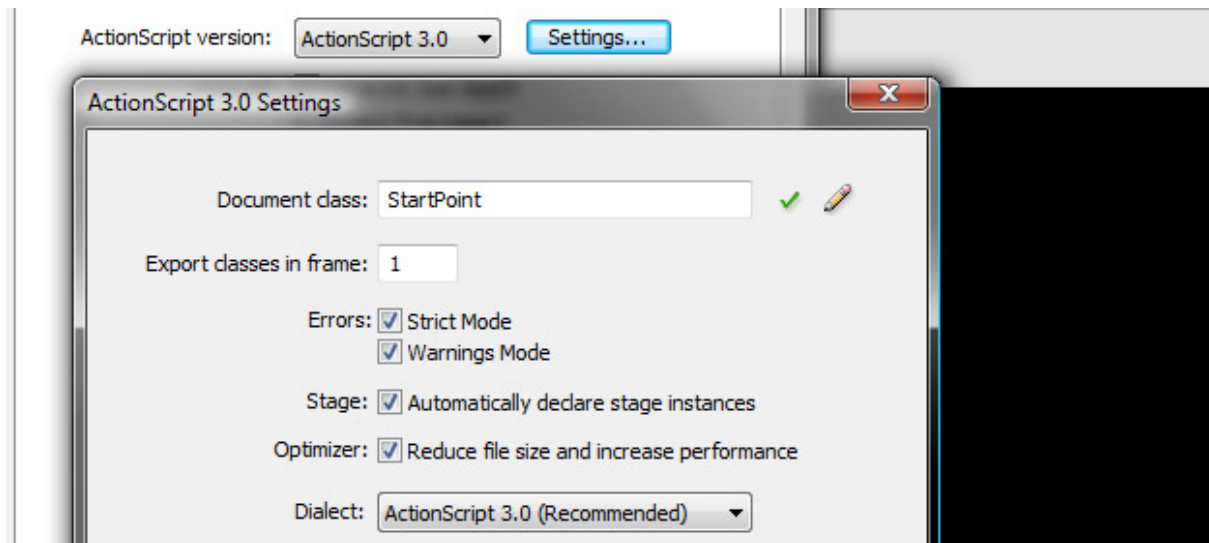
In the **fla**, go to the **Property Inspector**.



Type in the name of the Document Class in the field, do **not** include the extension (no .as). If it were in a subfolder, we would do the path using dots, e.g. **mySubFolder.StartPoint**



Hit ENTER. If it gives you an error, you have misspelt or not created or mis-linked the AS file. Check and fix.



NOTE: In the Property Inspector, click on Publish **Settings**.

In the Flash tab, beside the ActionScript version, click Settings again.

Here you will see information about which frame the class is exported at and how it responds to errors.

Anyway, back to doing something. Go to the StartPoint as file, and type in the following:

```
package{  
  
    import flash.display.*;  
  
    import flash.events.*;  
  
    class StartPoint extends MovieClip{  
  
        function StartPoint(){  
  
            trace("StartPoint started");  
  
        }//initialiser  
  
    }//class  
  
}//package
```

Save – always save your AS file before testing – and Test.

You get an error: *“ReferenceError: Error #1065: Variable StartPoint is not defined.”*

Variables need to be marked as private, public, or whatever. I will not get into that too much right now. We need to add **public** into our code, before the word ‘class’.

```
package{  
  
    import flash.display.*;  
  
    import flash.events.*;  
  
    public class StartPoint extends MovieClip{  
  
        function StartPoint(){  
  
            trace("StartPoint started");  
  
        }//initialiser  
  
    }//class  
  
}//package
```

When you test in now, all that should happen is it should trace the string we asked it to. Not much, but it is a start. I find I prefer to indicate a trace like that into all my AS files so I can check if they are running, or at what point they execute.

What you have covered here will form the base of ALL classes, packages or AS files you create in the future.

In Part 2: Variables, functions, ‘object’ classes and mouse events.